НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет
Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:
Завідувач кафедри
_____ Олександр Коваль
«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційні технології моніторингу довкілля»

спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: «Програмні засоби доступу до хмарних середовищ розподілених електронних ресурсів»

Виконав (-ла):
студент (-ка) IV курсу, групи ТМ-61
Семенюк Максим Віталійович _____

Керівник:
Старший викладач,
Колумбет Вадим Петрович _____

Рецензент:
Доц., доц. каф. АЕС і ІТФ, к. т. н.,
Баранюк Олександр Володимирович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний
Кафедра автоматизації проектування енергетичних процесів і систем
Рівень вищої освіти перший рівень
Напрям підготовки 122 Комп’ютерні науки та інформаційні технології
Спеціалізація Інформаційні технології моніторингу довкілля

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту

_____Семенюк Максим Віталійовчи_____
(прізвище, ім’я, по батькові)

1. Тема роботи Програмні засоби доступу до хмарних середовищ розподілених електронних ресурсів

керівник роботи_____Колумбет Вадим Петрович, ст. Викладач_____
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25” травня 2020р. № **1168-с**

2. Строк подання студентом роботи ___15.06.2020 року_____
3. Вихідні дані до роботи ___Visual Studio 2019, C#, Razor, Google Drive API_____
4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)_____проаналізувати проблеми сучасних хмарних середовищ; розробити веб-сервіс для інтеграції з хмарним середовищем; надати інтерфейс взаємодії з хмарним середовищем;
5. Перелік ілюстративного матеріалу
Актуальність; мета; засоби реалізації; реалізація доступу до хмарного середовища в програмному продукту; зберігання облікових даних користувача на сервері; зберігання документів; функціонал сервісу; інтерфейс користувача; висновки
6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ”_8_”____січня_____2020__ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	08.01.2020	
2.	Вивчення та аналіз задачі	01.03.2020	
3.	Розробка архітектури та загальної структури системи	01.04.2020	
4.	Розробка структур окремих підсистем	15.-30.04.2020	
5.	Програмна реалізація системи	03.05.2020	
6.	Оформлення пояснювальної записки	10-20.05.2020	
7.	Захист програмного продукту	08.06.2020	
8.	Передзахист	08.06.2020	
9.	Захист	15.06.2020	

Студент _____ ___Семенюк М. В_____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _Колумбет В. П._____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Метою цієї роботи є створення програмного засобу доступу до хмарного середовища розподілених електронних ресурсів.

Систему можна використовувати як модуль програми або проекту.

Загальний обсяг роботи: 55 сторінка, 8 ілюстрацій та 8 бібліографічних найменувань.

Ключові слова: хмарне сховище, проектування, бібліотека класів, додаток.

ABSTRACT

The purpose of this work is to create a software tool for access to the cloud environment of distributed electronic resources.

The system can be used as a program or project module.

Total volume of work: 55 pages, 8 illustrations and 8 bibliographic titles.

Key words: cloud storage, design, class library, application.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

SOLID	— принципи єдиної відповідальності, відкритості / закритості, підстановки Барбари Лісков, поділу інтерфейсу і інверсії залежностей)
SaaS	— програмне забезпечення як послуга.
IaaS	— інфраструктура як послуга.
PaaS	— платформа як послуга.
CLI	— загальна мовна інфраструктура.
IIS	— інтернет-сервер інформації

ВСТУП

Хмарне середовище - це модель обслуговування, в якій дані передаються та зберігаються в системах віддаленого зберігання, де вони підтримуються, управляються, створюються резервними копіями та стають доступними для користувачів через мережу (як правило, в Інтернеті). Зазвичай користувачі платять за хмарне зберігання даних щомісячно. Незважаючи на те, що вартість гігабайт даних була радикально знижена, постачальники хмарних сховищ додали операційні витрати, які можуть зробити цю технологію значно дорожчою. Безпека хмарних сервісів зберігання продовжує залишати занепокоєння серед користувачів. Постачальники послуг намагаються усунути ці страхи, підвищивши свої можливості безпеки, включивши в свої послуги шифрування даних, багатофакторну автентифікацію та покращену фізичну безпеку. Хмарне сховище може бути використане для копіювання образу віртуальної машини із хмари на локальне місце, або імпортування образу віртуальної машини з локального місця в бібліотеку хмари. Крім того, хмарні сховищі можуть використовуватись, щоб переміщувати образи віртуальних машин між обліковими записами користувачів або між центрами обробки даних

Існує три основні хмарні моделі доступу до пам’яті: державна, приватна та гібридна.

Державна модель зберігання у хмарі забезпечують середовище зберігання для багатьох орендарів, яке найбільше підходить для неструктурованих даних на основі підписки. Дані зберігаються в центрах обробки даних провайдерів послуг із збереженням даних, що поширюються по кількох регіонах або континентах. Клієнти, як правило, платять на основі користування, аналогічно моделі оплати комунальних послуг, у багатьох випадках також є плата за транзакції на основі частоти та обсягу даних до яких можна отримати доступ. У цьому ринковому секторі переважають служба простого зберігання Amazon, Google Cloud Storage та Microsoft Azure.

Служба приватного хмарного зберігання надається внутрішніми ресурсами зберігання, розгорнутими як виділене середовище, захищене між брандмауером організації. Вбудовані у внутрішні мережі приватні хмарні сховища імітують деякі функції комерційно доступних публічних хмарних сервісів, забезпечуючи простий доступ та розподілення ресурсів зберігання для бізнес-користувачів, а також протоколи зберігання об'єктів. Приватні хмари підходять для користувачів, які потребують налаштування та більшого контролю над своїми даними або мають суворі вимоги щодо безпеки даних чи регуляторних норм.

Гібридна хмара - це поєднання приватного хмарного сховища та сторонніх публічних служб зберігання хмари, щоб оперативно інтегрувати дві платформи. Ця модель пропонує гнучкість бізнесу та більше можливостей розгортання даних. Наприклад, організація може зберігати активно використовувані та структуровані дані у локальній хмарі, а неструктуровані та архівні дані - у відкритій хмарі. Гібридне середовище також полегшує управління сезонними або непередбачуваними стрибками у створенні даних або доступі шляхом "хмарного вибуху" до зовнішньої служби зберігання та уникає необхідності додавати внутрішні ресурси зберігання. В останні роки спостерігається посилення прийняття гібридної хмарної моделі. Незважаючи на свої переваги, гібридна хмара представляє технічні, ділові та управлінські проблеми. Наприклад, приватні робочі навантаження повинні отримувати доступ та взаємодіяти з громадськими постачальниками хмарних сховищ, тому сумісність та надійне, достатнє підключення до мережі є дуже важливими факторами. Система хмарного зберігання на рівні підприємства повинна бути масштабованою відповідно до поточних потреб, доступною з будь-якого місця та додатково-агностичною.

Характеристики хмарного зберігання

Хмарне сховище базується на віртуалізованій інфраструктурі з доступними інтерфейсами, майже миттєвою еластичністю, масштабованістю та дозованими ресурсами. Дані, що базуються на хмарі, зберігаються в логічних пулах на різних

розрізнених товарних серверах, розташованих у приміщеннях або в центрі обробки даних, якими керується сторонній постачальник хмарних послуг. Використовуючи RESTful API, протокол зберігання об'єктів, зберігає файл та пов'язані з ним метадані як єдиний об'єкт та присвоює йому ідентифікаційний номер. Коли вміст потрібно отримати, користувач представляє ідентифікатор системі та вміст збирається з усіма своїми метаданими, автентифікацією та безпекою.

В останні роки постачальники об'єктів зберігання додали функції та функції файлової системи до свого програмного забезпечення та обладнання для зберігання об'єктів значною мірою тому що зберігання об'єктів не було прийнято досить швидко. Наприклад, хмарний шлюз для зберігання даних може забезпечити передній кінець емуляції файлової системи до їх об'єктного сховища, таке розташування часто дозволяє програмам отримувати доступ до даних без фактичної підтримки протоколу зберігання об'єктів. Усі програми резервного копіювання використовують протокол зберігання об'єктів, що є однією з причин того, що резервне копіювання в хмарний сервіс було початковою успішною програмою для хмарного зберігання.

Записка містить 5 розділів.

У першому розділі описується завдання створення програмного засобу доступу та редагування реєстру в хмарному сховищі.

У другому описується аналіз проблеми хмарного сховища.

У третьому розділі вказуються основні засоби розробки даної системи.

У четвертому розділі дано опис реалізованого програмного продукту та його архітектури.

У п'ятому розділі описано роботу користувача з програмною системою.

1 ЗАВДАННЯ СТВОРЕННЯ ПРОГРАМНОГО ЗАСОБУ ДОСТУПУ ДО ХМАРНИХ СЕРЕДОВИЩ РОЗПОДІЛЕНИХ ЕЛЕКТРОННИХ РЕСУРСІВ

Метою розробки системи доступу до хмарних середовищ розподілених електронних ресурсів є створення програмного продукту, що надає простий інтерфейс користувачу для роботи з хмарним середовищем та в майбутньому використовувати як модуль будь-якого проекту.

1.1 Мета створення програмного засобу доступу до хмарного середовища розподілених електронних ресурсів

Метою розробки системи доступу до хмарних середовищ розподілених електронних ресурсів є створення програмного продукту, що надає простий інтерфейс користувачу для роботи з хмарним середовищем.

В основу закладений сервіс яким можуть користуватися будь-який користувач в якого є доступ до хмарного середовища. Сервіс можна використовувати як модуль будь-якого проекту при невеликій кастомізації до програмного продукту або програмного рішення.

1.2 Вхідні дані

Розроблена програма потребує лише дані користувача в хмарному середовищі.

Від користувача не вимагається ніяких додаткових вхідних даних. Достатньо лише завантажити json файл, який містить інформацію, яка потрібна для аутентифікації та авторизації користувача в хмарному середовищі. Після він може отримати свої дані (файли, папки, директорії).

Завдяки невеликій кількості вхідної інформації розроблюваний додаток не має викликати дискомфорту у користувача під час його використання.

1.3 Компоненти системи

В результаті розробки програмного продукту, користувач матиме змогу створювати нові файли та директорії, видаляти, додавати та корегувати файли.

Розроблена система повинна забезпечувати наступні можливості:

- доступ до хмарного середовища;
- просту взаємодію з клієнтом;
- моделювання репозиторію;
- моделювання файлової системи;
- вивід реєстру хмарного середовища та можливість завантажити на свій носій;
- редагування реєстру;

1.4 Потенційні користувачі

Система, розроблена у ході виконання цієї бакалаврської дипломної роботи може бути корисна для людей, яким потрібна інтеграція з хмарним середовищам в проекті. Тобто систему можна використовувати як модуль будь-якого проекту. Також система буде корисна для швидкого доступу до хмарного середовища.

2 АНАЛІЗ ПРОБЛЕМИ ХМАРНОГО СЕРЕДОВИЩА

Ми знаємо, що хмарні обчислення - це набір «віддаленої обробки», який виконується віддаленим сервером, доступний у будь-який час і в будь-якому місці.

2.1 Категорії віддаленої обробки

Щоб розібратися з ризиками використання хмарного середовища потрібно знати категорії (етапи) зберігання даних в хмарному середовищі. Як правило майже всі хмарні середовища містять три основі категорії:

- Програмне забезпечення як послуга (SaaS) - програмне забезпечення, постачається та управляється віддалено одним або кількома постачальниками. Для початку Software-as-a-Service або SaaS - це популярний спосіб доступу та оплати за програмне забезпечення. Замість того, щоб встановлювати програмне забезпечення на власних серверах, компанії SaaS дозволяють орендувати розміщене програмне забезпечення, як правило, щомісячна або щорічна плата за підписку. Все більше і більше інструментів, пов'язаних з управлінням CRM, маркетингом та фінансами, використовують бізнес-аналітику SaaS та технології, і навіть Adobe Creative Suite прийняла цю модель.
- Інфраструктура як послуга (IaaS) - обчислювальні ресурси, доповнені можливостями зберігання, розміщення постачальників та доступними клієнтам за запитом.
- Платформа як послуга (PaaS) - широка колекція прикладних інфраструктурних сервісів. Ці послуги включають платформу додатків, інтеграцію, управління бізнес-процесами та сервіси баз даних.

Немає сумнівів, що хмарне середовище забезпечує різноманітні типи рішень для зберігання даних, кожне з яких має свої обмеження та переваги.

2.2 Загрози та проблеми зберігання хмарних даних

Можна стверджувати, що зберігання даних є найважливішим компонентом хмарних обчислень, і саме тому безпека даних над розподіленими обчисленнями завжди є великою проблемою. Є багато питань безпеки щодо зберігання хмарних даних; нижче наведені деякі найбільш поширені та серйозні:

2.2.1 Анонімність

У хмарних сховищах даних анонімність - це метод затемнення опублікованих даних та ключової інформації, що запобігає асоційованій особі власника даних. Однак анонімність даних має різні вразливості, такі як прихована ідентичність супротивних загроз, лазівки в процедурі повторної ідентифікації.

2.2.2 Викрадення трафіку облікових записів або служб

Зазвичай він виконується за допомогою викрадених облікових даних і залишається найбільшою загрозою. За допомогою викрадених облікових даних зловмисники часто можуть отримувати доступ до критичних областей розгорнутих хмарних обчислювальних служб, що дозволяє їм порушувати конфіденційність, цілісність та доступність цих служб. Різні зловмисні методи, такі як використання вразливих програм, таких як крадіжка паролів / облікових даних та атаки переповнення буфера, а також фішинг-атаки можуть призвести до втрати контролю над обліковими записами користувачів. Це може призвести до того, що хакер отримає контроль над обліковим записом користувача та підслуховує транзакції, маніпулює даними та / або перенаправляє клієнтів на веб-сторінку конкурента або на невідповідні сайти.

2.2.3 Втрата даних та витік даних

Порушення даних є результатом нав'язливої дії, і втрата даних може статися, коли диск накопичується, не створюючи резервної копії. Це втрата конфіденційності, довіри та має прямий вплив на політику угоди про обслуговування, яка є основною проблемою користувачів хмари.

2.2.4 Криптографія

Часто криптографічні механізми, здається, не спрацьовують, коли застосовується захід безпеки. У хмарі застосовуються криптографічні методи для подолання лазів в зонах безпеки. Однак такі виклики, як головна факторизація великої кількості в RSA та дискретні логарифмічні проблеми в ECC та неправильне виконання, можуть спричинити грубу атаку. Погане управління ключами, ефективність обчислень, перевірені дані - це також інші проблеми, пов’язані з хмарною криптографією.

2.2.5 Проблеми цілісності та конфіденційності

Цілісність є найважливішим елементом інформаційної системи для захисту даних від несанкціонованого зміни, видалення чи зміни. Проблема безпеки виникає, коли неправильно визначені параметри безпеки. Хмара з багато орендарів може призвести до порушення цілісності та конфіденційності, тому збільшується кількість користувачів, що підвищують ризик безпеки. Деякі відомі напади цілісності включають атаку людини на середині, викрадення сеансу, атаку збиття даних, пароль, обнюхування пакетів та атаки соціального інжинірингу і можуть серйозно вплинути на конфіденційність інформації.

2.2.6 Зловмисне програмне забезпечення

Це стосується нападу електронних злочинів для введення зловмисного програмного забезпечення у хмарне сховище під назвою "Botnets", перетворення їх на "зомбі", спрямоване на атаку більшості комп'ютерних мережевих серверів.

2.3 Висновки

Хмарні середовища пропонують безліч переваг, за потребою, масштабованість, ресурси та рішення на базі IT, без необхідності вкладати гроші в нову інфраструктуру. Незважаючи на ці особливості, безпека зберігання є критичним моментом, який стоїть перед цією технологією.

3 ЗАСОБИ РОЗРОБКИ

Одним із важливіших завдань є вибір технологій для вирішення тих чи інших проблем при написані програмного продукту. Проект був розроблений за допомогою середовищу розробки програмних додатків Microsoft Visual Studio та платформою .NET.

Для розробки сервісу використовувалась об’єктно-орієнтована мова програмування C#.

Для доступу та редагуванню реєстру хмарного середовища, було використано сервіс від компанії Google – Google Drive API, в якого є базові методи для взаємодії з хмарним середовищем Google Disk.

3.1 Платформа .NET Framework

Платформа .Net - це платформа розробки програмного забезпечення, розроблена Microsoft. Рамка призначена для створення додатків, які працюватимуть на платформі Windows. Перша версія фреймворку .Net була випущена в 2002 році.

Версія отримала назву .Net Framework 1.0. З цього часу фреймворк .Net пройшов довгий шлях, а поточна версія - 4.7.1.

Платформа .Net може використовуватися для створення як програмних, так і веб-програм. Веб-сервіси також можуть бути розроблені за допомогою .Net Framework.

Платформа також підтримує різні мови програмування, такі як Visual Basic і C #. Тож розробники можуть вибирати мову для розробки потрібної програми.

Основна архітектура .Net фреймворку наведена нижче (Рисунок 3.1.1)

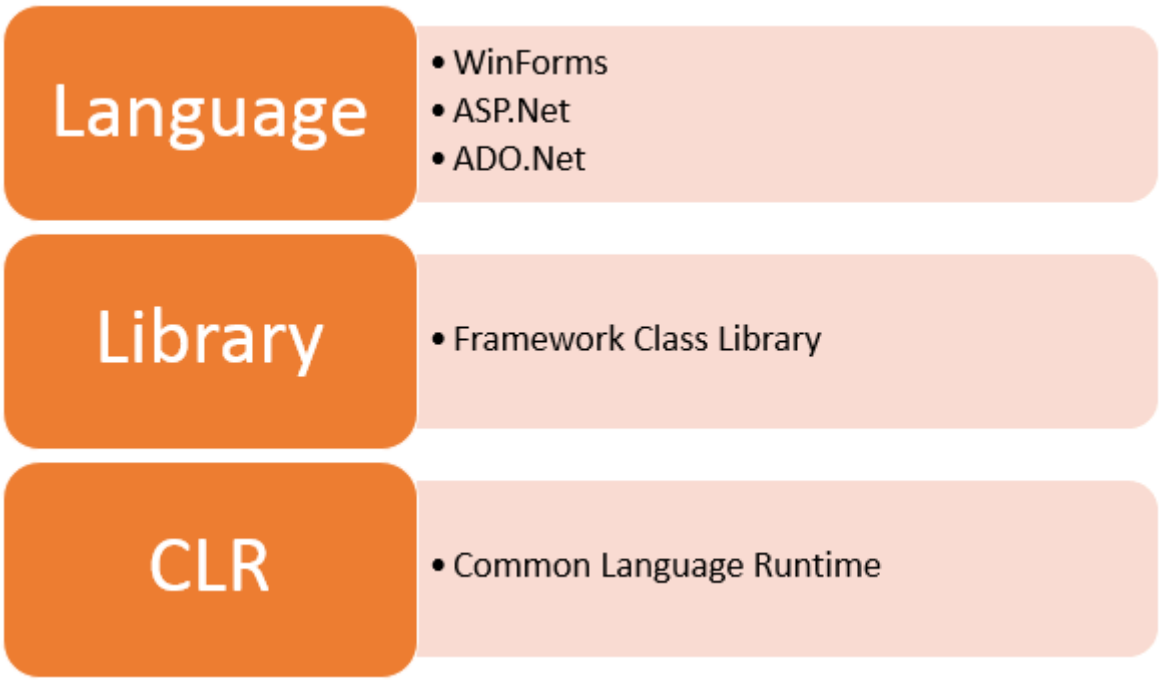

Рисунок 3.1.1 — Основна архітектура .Net фреймворку

Архітектура .Net Framework базується на наступних ключових компонентах;

3.1.1 Загальна мовна інфраструктура

"Загальна мовна інфраструктура" або CLI - це платформа, на якій виконуються програми .Net.

CLI має такі основні характеристики:

1. Обробка винятків - помилки, які виникають під час виконання програми.

Приклади винятків:

- Якщо програма намагається відкрити файл на локальній машині, але файл відсутній.
- Якщо програма намагається отримати деякі записи з бази даних, але підключення до бази даних недійсне.

2. Збір сміття - це процес видалення небажаних ресурсів, коли вони більше не потрібні.

Приклади збору сміття є:

- Ручка файлу, яка більше не потрібна. Якщо програма закінчила всі операції над файлом, обробка файлів більше не потрібна.
- Підключення до бази даних більше не потрібнf. Якщо програма закінчила

всі операції над базою даних, можливо, більше не потрібно буде з'єднання з базою даних.

3. Робота з різними мовами програмування -

Розробник може розробляти додаток на різних мовах програмування .Net.

- Мова - перший рівень - це сама мова програмування, найпоширеніші - VB.Net і C #.
- Компілятор - є компілятор, який буде окремим для кожної мови програмування. Отже, що лежить в основі мови VB.Net, буде окремий компілятор VB.Net. Так само і для C # у вас буде інший компілятор.
- Перекладач загальної мови - це останній рівень в .Net, який би використовувався для запуску програми .Net, розробленої на будь-якій мові програмування. Отже наступний компілятор відправить програму на рівень CLI для запуску програми .Net.

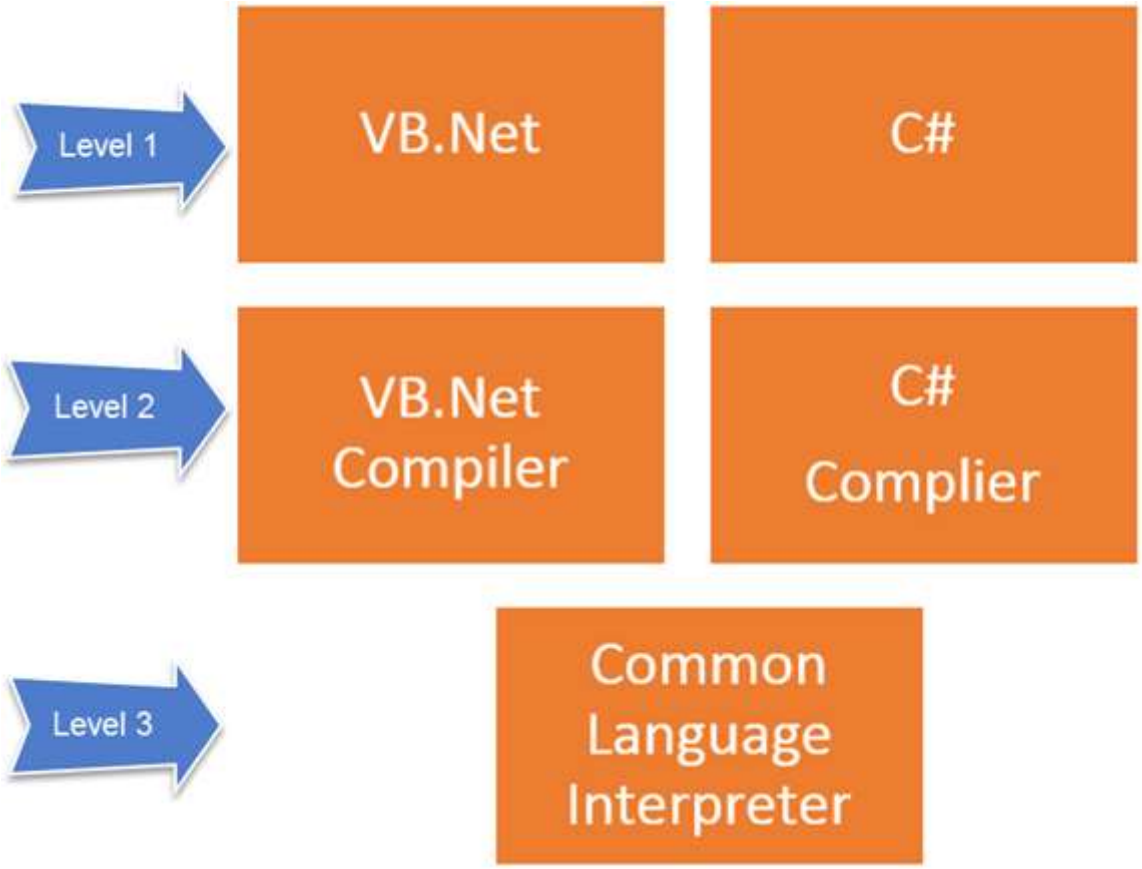

Рисунок 3.1.2 — Загальна мовна інфраструктура

3.1.2 Бібліотека класів

.NET Framework включає набір бібліотек стандартних класів.

Бібліотека класів - це сукупність методів та функцій, які можна використовувати для основної мети. Наприклад, є бібліотека класів з методами для обробки всіх операцій на рівні файлів. Отже, існує метод, який можна використовувати для читання тексту з файлу. Так само існує метод запису тексту у файл.

Більшість методів розділені на простори імен System. * Або Microsoft. *. (Зірочка * просто означає посилання на всі методи, що належать до простору імен системи або Майкрософт). Простір імен - це логічне розділення методів.

3.1.3 Мови

Типи додатків, які можна вбудувати в .Net Framework, класифікуються широко в наступні категорії:

- WinForms - використовується для розробки додатків на основі форм, які працюватимуть на машині кінцевого користувача. Блокнот - це приклад клієнтської програми.
- ASP.Net - використовується для розробки веб-додатків, які запускаються в будь-якому веб-переглядачі, наприклад Internet Explorer, Chrome або Firefox. Веб-додаток оброблятиметься на сервері, на якому встановлено Інтернет-інформаційні послуги. Internet Information Services або IIS - це компонент Microsoft, який використовується для виконання програми ASP.Net. Потім результат виконання надсилається на клієнтські машини, а вихід відображається в браузері.
- ADO.Net - технологія використовується для розробки програм які взаємодіють з базами даних, такими як Oracle або Microsoft SQL Server.

Microsoft завжди гарантує, що рамки .Net відповідають усім підтримуваним операційним системам Windows.

3.1.4 Принципи дизайну .Net Framework

Наступні принципи дизайну .Net Framework - це те, що робить його дуже актуальним для створення додатків на базі .Net.

Інтероперабельність - .Net-рамка забезпечує велику підтримку назад. Припустимо, якщо у вас був додаток, побудований на старій версії рамки .Net, скажімо 2.0. А якщо ви спробували запустити ту саму програму на машині, яка мала вищу версію. Додаток все ще запрацює. Це пояснюється тим, що з кожним випуском Microsoft гарантує, що старіші рамкові версії добре поєднуються з останньою версією.

Переносимість. Програми, побудовані на основі .Net, можна зробити для роботи на будь-якій платформі Windows. І ось останнім часом Microsoft також передбачає, щоб продукти Microsoft працювали на інших платформах, таких як iOS та Linux.

Безпека - .NET Framework має хороший механізм захисту. Вбудований механізм захисту допомагає як для перевірки, так і для перевірки програм. Кожна програма може чітко визначити свій механізм захисту. Кожен механізм захисту використовується для надання користувачеві доступу до коду або до запущеної програми.

Управління пам'яттю - Загальна мова виконання виконує всю роботу або управління пам'яттю. Рамка .Net має всі можливості бачити ті ресурси, які не використовується запущеною програмою. Тоді вони б відповідно звільнили ці ресурси. Це робиться за допомогою програми під назвою "Збір сміття", яка працює в рамках .Net.

Збір сміття працює через регулярні проміжки часу і постійно перевіряє, які системні ресурси не використовуються, і звільняє їх відповідно.

Спрощене розгортання - .Net Framework також має інструменти, які можна використовувати для упаковки програм, побудованих на .Net Framework. Потім ці пакунки можуть бути розповсюджені на клієнтських машинах. Потім пакети автоматично встановлюють додаток.

3.2 Середовище розробки Microsoft Visual Studio 2019

Microsoft Visual Studio 2019 Community – продукт від фірми Майкрософт, яка включає інтегроване середовище розробки програмного забезпечення та низку інших інструментальних засобів. Продукт дозволяють розробляти як консольні програми, так і програми з графічним інтерфейсом. Для вирішення завдань доступу та редагування реєстру хмарного середовища використовується технологія ASP.NET.

Середовище Visual Studio дозволяє розробляти додатки, використовуючи різні мови програмування: Visual C#, Visual Basic, Visual F#, Visual C++, Python і т.д. Також існує можливість розробляти не тільки під Windows, а і під інші популярні платформи: Android, IOS.

Версія Visual Studio Community є абсолютно безкоштовно для учнів, студентів та розробників програм з відкритим програмним кодом. В новій версії Visual Studio 2019 також доступні нові можливості.

3.2.1 Інтерфейс користувача

Коли ви відкриваєте Visual Studio, існує ряд вікон інструментів, які дозволяють взаємодіяти з вашим кодом: Вікна інструментів Visual Studio (Рисунок 3.2.1)

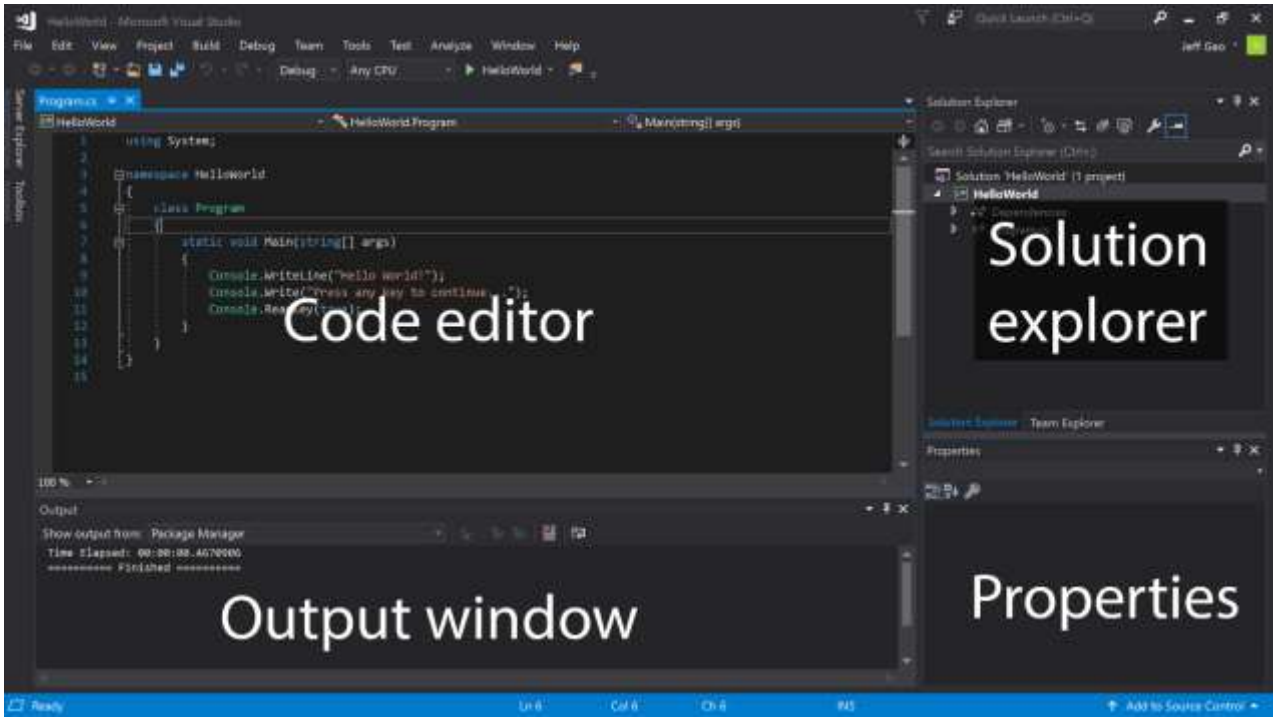

Рисунок 3.2.1 - Вікно інструментів

- Code editor - панель для написання коду.
- Solution explorer - показує файли, з якими ви працюєте.
- Properties - надає додаткову інформацію та контекст щодо вибраних частин вашого проекту.
- Output window - відображаються повідомлення про налагодження та помилки, попередження компілятора, повідомлення про стан та інші результати.

У верхній частині екрана - меню Visual Studio, яка використовується для запуску різних команд. Ось найважливіших з них (Рисунок 3.2.2).

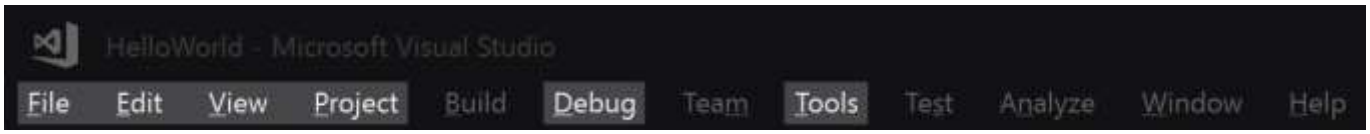

Рисунок 3.2.2 - Меню

- File - містить команди для створення, відкриття та збереження проектів.
- Edit - містить команди для пошуку, зміни та повторного змінення коду.
- View - ви відкриваєте додаткові вікна інструментів у Visual Studio.
- Project - дозволяє додавати файли та залежності в проект.
- Debug - містить команди для запуску коду та використання функцій налагодження.
- Tools - містить команди для зміни налаштувань, додавання функціональних можливостей Visual Studio через розширення та доступу до різних інструментів Visual Studio.

Панель інструментів Visual Studio, показана нижче меню, забезпечує швидкий доступ до найпоширеніших команд.

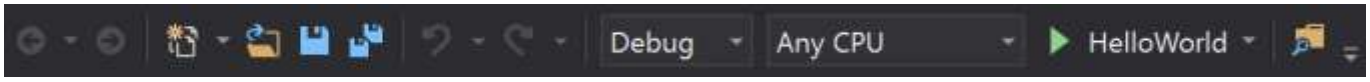

Рисунок 3.2.3 — Панель інструментів

3.2.2 Огляд нових можливостей

Розробка програмних рішень за допомогою Microsoft Visual Studio надає ряд переваг такі як:

- Розробка. Дозволяє зосередитись на одному, а також підвищує продуктивність завдяки оптимізації, можливості миттєвої очистки коду і більш точнішим результатом пошуку.
- Спільна робота. Можна скористатися можливостями спільної роботи в рамках робочого процесу Git-first, функціями редагування та налагодження, а також рецензування коду прямо в Visual Studio.
- Налагодження. Оптимізація використаної пам'яті і створення автоматичних моментальних знімків при виконанні програми.

3.3 Веб-інтерфейс

Веб-інтерфейс — це сукупність засобів, за допомогою яких користувач взаємодіє з веб-сайтом або веб-застосунком через браузер. Веб-інтерфейси отримали широке поширення у зв'язку із зростанням популярності всесвітньої павутини і відповідно повсюдного розповсюдження веб-браузерів.

Однією з основних вимог до веб-інтерфейсів є їх однаковий зовнішній вигляд і однакова функціональність при роботі в різних браузерах.

Класичним і найпопулярнішим методом створення веб-інтерфейсів є використання HTML із застосуванням CSS і JavaScript, як правило за допомогою скриптових мов на стороні сервера. Проте різна реалізація HTML, CSS, DOM і інших специфікацій в браузерах викликає проблеми при розробці веб-застосунків і їхньої подальшої підтримки. Крім того, можливість користувача настроювати багато параметрів браузера (наприклад, розмір шрифту, кольору, відключення підтримки сценаріїв) може перешкоджати коректній роботі інтерфейсу.

З розвитком DHTML та JavaScript набув популярності підхід до розробки інтерфейсної частини веб-застосунків, названий AJAX (асинхронний JavaScript). Серцем технології є здатність веб-сторінки зініціювати запит до веб-сервера і отримати потрібні дані, так щоб інтерфейс не перезавантажував сторінку цілком, а лише довантажують необхідні дані і змінив потрібні частини сторінки, що робить їх більш інтерактивними і продуктивними.

Веб-інтерфейси зручні тим, що дають можливість вести спільну роботу співробітникам, які не перебувають в одному офісі (наприклад, веб-інтерфейси часто використовуються для заповнення різних баз даних або публікації матеріалів в інтернет-ЗМІ).

Добрим прикладом використання і віддачі веб-інтерфейсу є Вікіпедія: практично весь вміст вільної всесвітньої енциклопедії створений і доданий на сторінки сайту за допомогою веб-інтерфейсу.

Веб-інтерфейс дає можливість універсального віддаленого доступу до служб та пристроїв, у цьому технології практично нема альтернатив. Але водночас, оскільки такий інтерфейс доступний усім, постають серйозні питання забезпечення безпеки, зокрема автентифікація та авторизація користувачів, шифрування переданих даних від сторонніх очей тощо.

3.4 Платформа ASP.NET MVC Framework

ASP.NET MVC Framework - платформа для створення веб-додатків, який реалізує шаблон Model-view-controller (про MVC в наступній главі).

Основні компоненти ASP.NET MVC:

Платформа ASP.NET MVC базується на взаємодії трьох компонентів: контролера, моделі та представлення. Контролер приймає запити, обробляє користувача введення, взаємодіє з моделлю і представленням і повертає користувачеві результат обробки запиту.

Модель представляє шар, що описує логіку організації даних в додатку. Подання отримує дані з контролера і генерує елементи призначеного для користувача інтерфейсу для відображення інформації. Для взаємодії з користувачем було додано такі технології двигун уявлень та маршрутизацію:

Двигун уявлень:

Для управління розміткою і вставками коду в поданні використовується движок уявлень. До версії MVC 5 використовувалися два двигуна: Web Forms і Razor.

Починаючи з MVC 5 єдиним двигуном, вбудованим за замовчуванням, є Razor. Движок WebForms використовує файли .aspx, а Razor - файли .cshtml і .vbhtml для зберігання коду уявлень. Основою синтаксису Razor є знак @, після якого здійснюється перехід до коду на мовах C # / VB.NET [26]. Також можливо і використання сторонніх движків. Файли уявлень не є стандартними статичними сторінками з кодом html, а в процесі генерації контролером відповіді з використанням уявлень компілюються в класи, з яких потім генерується сторінка html.

Маршрутизація:

При обробці запитів фреймворк ASP.NET MVC спирається на систему маршрутизації, яка зіставляє всі вхідні запити з певними в системі маршрутами, які вказують який контролер і метод повинен обробити такий запит. Вбудований маршрут за умовчанням передбачає триланкову структуру: контролер / дію / параметр.

3.5 Висновки

Для розробки було встановлено середовище Microsoft Visual Studio 2017 з інтегрованою платформою розробки .NET Framework 4.6 і мовою програмування C# 8.0. Для використання інструментів для побудови розширень до середовища розробки було встановлено комплект розробки програмного забезпечення із усіма відповідними бібліотеками і шаблонами.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ ДОСТУПУ ДО ХМАРНОГО СЕРЕДОВИЩА РОЗПОДІЛЕНИХ ЕЛЕКТРОННИХ РЕСУРСІВ

В даному розділі міститься пояснення архітектурного рішення при розробці програмного хасобу доступу до хмарних середовищ розподілених електронних ресурсів.

4.1 Паттерн MVC

Модель–представлення–контролер — архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення.

Шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування.

Моделі:

Містять або представляють дані, з якими працюють користувачі. Вони можуть бути простими моделями уявлень, які тільки представляють дані, що передаються між уявленнями і контролерами; або ж вони можуть бути моделями предметної області, які містять бізнес-дані, а також операції, перетворення і правила для маніпулювання цими даними.

Представлення:

Застосовуються для візуалізації деякої частини моделі у вигляді призначеного для користувача інтерфейсу.

Контролери:

Обробляють запити, виконують операції з моделлю і вибирають уявлення для візуалізації користувачеві.

Основна мета застосування цієї концепції полягає в розмежуванні бізнес-логіки (моделі) від її візуалізації (уявлення, виду). За рахунок такого поділу підвищується можливість повторного використання коду. Найбільш корисне застосування даної концепції в тих випадках, коли користувач повинен бачити ті ж самі дані одночасно в різних контекстах і / або з різних точок зору. Зокрема, виконуються наступні завдання:

- До однієї моделі можна приєднати кілька видів, при цьому не зачіпаючи реалізацію моделі. Наприклад, деякі дані можуть бути одночасно представлені у вигляді електронної таблиці, гістограми і кругової діаграми;
- Не торкаючись реалізацію видів, можна змінити реакції на дії користувача (натискання мишею на кнопці, введення даних) - для цього досить використовувати інший контролер;
- Ряд розробників спеціалізується тільки в одній з областей: або розробляють графічний інтерфейс, або розробляють бізнес-логіку. Тому можливо добитися того, що програмісти, які займаються розробкою бізнес-логіки (моделі), взагалі не будуть обізнані про те, яке уявлення буде використовуватися.

4.2 Паттерн SOLID

SOLID - це абревіатура п'яти основних принципів проектування в об'єктно-орієнтованому програмуванні - Single responsibility, Open-closed, Liskov substitution, Interface segregation і Dependency inversion. У перекладі на українську: принципи єдиної відповідальності, відкритості / закритості, підстановки Барбари Лісков, поділу інтерфейсу і інверсії залежностей.

Абревіатура SOLID була запропонована Робертом Мартіном, автором кількох книг, широко відомих в співтоваристві розробників. Ці принципи дозволяють будувати на базі ООП масштабовані і супроводжувані програмні продукти зі зрозумілою бізнес-логікою:

- Single responsibility - принцип єдиної відповідальності
- Open-closed - принцип відкритості / закритості
- Liskov substitution - принцип підстановки Барбари Лісков
- Interface segregation - принцип поділу інтерфейсу
- Dependency inversion - принцип інверсії залежностей

Принцип єдиною обов'язки / відповідальності (single responsibility principle / SRP) позначає, що кожен об'єкт повинен мати один обов'язок і цей обов'язок повинен бути повністю інкапсульований в клас. Всі його сервіси повинні бути спрямовані виключно на забезпечення цього обов'язку.

Принцип відкритості / закритості (open-closed principle / OCP) декларує, що програмні сутності (класи, модулі, функції і т. п.) Повинні бути відкриті для розширення, але закриті для зміни. Це означає, що ці сутності можуть міняти свою поведінку без зміни їх вихідного коду.

Принцип підстановки Лісков (Liskov substitution principle / LSP) в формулюванні Роберта Мартіна: «функції, які використовують базовий тип, повинні мати можливість використовувати підтипи базового типу не знаючи про це».

Принцип поділу інтерфейсу (interface segregation principle / ISP) в формулюванні Роберта Мартіна: «клієнти не повинні залежати від методів, які вони не використовують». Принцип поділу інтерфейсів говорить про те, що занадто «товсті» інтерфейси необхідно розділяти на більш дрібні і специфічні, щоб клієнти маленьких інтерфейсів знали тільки про методи, які необхідні їм у роботі. У підсумку, при зміні методу інтерфейсу не повинні змінюватися клієнти, які цей метод не використовують.

Принцип інверсії залежностей (dependency inversion principle / DIP) - модулі верхніх рівнів не повинні залежати від модулів нижніх рівнів, а обидва типи модулів повинні залежати від абстракцій; самі абстракції не повинні залежати від деталей, а ось деталі повинні залежати від абстракцій.

4.3 Файлове середовище Google Drive

Google Drive (Google Disk) - сховище даних, яке належить компанії Google Inc., що дозволяє користувачам зберігати свої дані на серверах у хмарі і ділитися ними з іншими користувачами в Інтернеті. Google Drive включає Google Документи, Таблиці та Презентації, офісний пакет, який дозволяє спільно редагувати документи, електронні таблиці, презентації, малюнки, форми, і багато іншого. Станом на жовтень 2014 року мав 240 мільйонів щомісячно активних користувачів.

Для синхронізації файлів між комп'ютером користувача и хмарним сховищем необхідне програмне забезпечення Google Drive ('клієнт') на комп'ютері користувача. Клієнт Взаємодіє з Google Drive так, щоб оновлення на одній стороні поширювались на іншу, тому, як правило, містять одні й ті ж дані.

Клієнтське програмне забезпечення Google Drive доступна для наступних пристроїв: ПК під управлінням Windows XP, Windows Vista, Windows 7 и Windows 8 з розділамі NTFS або Mac OS X 10.6 (Snow Leopard) або вище; Android смартфони та планшети з Android 2.1 (Eclair) и вище; iPhone та iPad з прошивкою 5.0 або вище.

Google Drive доступна в автономному режимі браузера Google Chrome через додаток Chrome, з веб-магазину Chrome. Документи, електронні табліці, презентації та малюнки можна переглядати та редагувати за допомогою автономних додатків Chrome.

4.3.1 Особливості

- Спільне користування

Google Drive включає в себе систему обміну файлами, де творець файлу або теки за замовчуванням є його власником. Власник має можливість регулювати видимість файлу або теки для інших користувачів. Право власності підлягає передачі. Файли і теки можуть спільно використовуватись приватно конкретними користувачами, що мають обліковий запис Google, використовуючи свої @gmail.com адреси електронної пошти. Для спільного використання файлів з користувачами, які не мають облікового запису Google потрібно зробити файли «доступними за посиланням». Це створює секретний URL для файлу, який може бути відкритий через

електронну пошту, блоги і т. д. Файли та теки можна зробити «загальнодоступними в Інтернеті», що означає, що вони можуть бути проіндексовані пошуковими системами і, таким чином, можуть бути знайдені і доступні будь-кому. Власник може також встановити рівень доступу. Три запропоновані рівня доступу, це «може редагувати», «може коментувати» і «може переглядати». Користувачі, що мають доступ для редагування можуть запрошувати інших редагувати.

- Сторонні додатки

Існує ряд зовнішніх веб-додатків («apps»), які працюють з Google Drive. Ці програми доступні у веб-магазині Chrome і сумісні з усіма підтримуваними браузерами. Щоб використовувати додаток, користувач повинен увійти в Chrome Web Store і встановити додаток. Деякі з цих додатків розроблені Google, наприклад, Google Docs, Sheets and Slides. Додатки Drive, які працюють з інтернет-файлами, використовуються для перегляду, редагування і створення файлів в різних форматах, редагування зображення та відео, факсу і підписування документів, управління проектами, створення блок-схем, додатків і т. д. Drive також може бути програмою за замовчуванням для обробки файлів у форматах, які ним підтримуються. Деякі з цих додатків також працюють в автономному режимі, але лише на Google Chrome і Chrome OS. Усі сторонні додатки можна встановити безкоштовно. Тим не менш, деякі з них стягують додаткову плату за продовження використання або доступ до додаткових функцій. Більшість додатків Drive мають дозвіл на доступ до файлів користувачів за межами Google Drive. Збереження даних з сторонніх додатків на Google Drive вимагає авторизації в перший раз. Google Drive SDK працює спільно з Google Drive UI і Chrome Web Store, щоб створити екосистему додатків, які можуть бути встановлені в Google Drive.

- Пошук

Результати пошуку можуть бути звужені за типом файлу, власником, видимістю і додатком для перегляду. Google Drive підтримує логічні оператори. За допомогою Google Goggles і технології оптичного розпізнавання символів (OCR), користувачі можуть шукати зображення за описом їх вмісту. Наприклад, пошук

«гори» покаже всі фотографії гір, а також будь-які текстові документи про гори. Пошук розпізнає текст в перших 100 сторінках текстових документів і текстових файлів PDF, і перші 10 сторінок PDF-файлів на основі малюнків. Текст у PDF-файлах і зображеннях видобувається за допомогою OCR.

- Доступність

25 червня 2014, Google анонсувала ряд оновлень в Google Drive, які мали на меті зробити сервіс більш доступним для користувачів з вадами зору. Це включає поліпшений доступ до клавіатури, підтримка масштабування, режим високої контрастності і кращої сумісності з зчитувачами екрана.

4.4 OAuth 2.0 протокол

Google API використовують протокол OAuth 2.0 для аутентифікації та авторизації. Google підтримує поширені сценарії OAuth 2.0, такі як веб-сервер, клієнтська установка та додатки пристроїв обмеженого введення.

Для початку потрібно отримати облікові дані клієнта OAuth 2.0 з консолі Google API. Потім клієнтська програма запитує маркер доступу з сервера авторизації Google, витягує маркер із відповіді та надсилає маркер API Google, до якого ви хочете отримати доступ.

Усі програми дотримуються основної схеми під час доступу до Google API Console за допомогою OAuth 2.0. На високому рівні виконується п'ять кроків:

1. Отримання облікових даних OAuth 2.0 з консолі Google API Console.

В Google API Console можна отримати облікові дані OAuth 2.0, такі як ідентифікатор клієнта та секрет клієнта, які відомі як Google, так і в програмі. Набір значень змінюється залежно від типу програми. Наприклад, програма JavaScript не вимагає секрету, але програма веб-сервера вимагає.

2. Отримання маркер доступу з сервера авторизації Google.

Перш ніж програма може отримати доступ до приватних даних за допомогою API Google, вона повинна отримати маркер доступу, який надає доступ до цього API. Один маркер доступу може надавати різний ступінь доступу до декількох API. Змінна

параметр, який називається область, керує набором ресурсів та операцій, які дозволяє маркер доступу. Під час запиту маркера доступу ваша програма надсилає одне або більше значень у параметрі області.

Існує кілька способів зробити цей запит, і вони залежать від типу програми. Наприклад, програма JavaScript може запитати маркер доступу за допомогою перенаправлення браузера на Google, тоді як програма, встановлена на пристрої, у якого немає браузера, використовує запити веб-служб.

Деякі запити потребують кроку аутентифікації, коли користувач входить у свій обліковий запис Google. Після входу в систему користувача запитують, чи готові вони надати один чи більше дозволів, про які запитує ваша програма. Цей процес називається згодою користувача.

Якщо користувач надає щонайменше один дозвіл, сервер авторизації Google надсилає програмі маркер доступу (або код авторизації, який програма може використовувати для отримання маркера доступу) та перелік областей доступу, наданих цим маркером. Якщо користувач не надає дозволу, сервер повертає помилку.

3. Управління доступом.

Потрібно порівняти області, що містяться у відповіді маркера доступу, з масштабами, необхідними для доступу до функцій та функцій програми, залежно від доступу до відповідного API Google.

Обсяг, включений у запит, може не відповідати обсягу, включеному у відповідь, навіть якщо користувач надав усі запитувані області. Потрібно звернутись до документації для кожного API Google щодо обсягів, необхідних для доступу. API може зіставити декілька значень рядка діапазону в одну область доступу, повертаючи ту саму рядок області для всіх значень, які дозволені в запиті.

4. Надсилання маркеру доступу в API.

Після того, як програма отримує маркер доступу, вона надсилає маркер API Google у заголовку запиту авторизації HTTP. Крім того, хороша практика REST уникати створення зайвих імен параметрів URI.

5. Оновлення маркеру доступу, якщо необхідно.

Токени доступу мають обмежений термін служби. Якщо програмі потрібен доступ до API Google за період дії одного маркера доступу, він може отримати маркер оновлення. Токен оновлення дозволяє вашій програмі отримувати нові маркери доступу.

Далі наведено ілюстрація взаємодії користувача з Google Servers (Рисунок 4.4.1)

Рисунок 4.4.1 - Взаємодії користувача з Google Servers

5 РОБОТА КОРИСТУВАЧА З СИСТЕМОЮ

В цьому розділі приведені системні вимоги для роботи з додатком та сценарії роботи користувача з ним.

5.1 Системні вимоги

Для забезпечення коректної та безвідмовної роботи програмного засобу доступу до хмарних середовищ потрібні мінімальні показники для роботи браузерів:

Процесор - intel Pentium 4 / Athlon 64 або пізнішої версії з підтримкою SSE2 .

Вільне місце на диску – 350 Мб.

Оперативна пам’ять – 512 Мб.

5.2 Робота користувача з програмним продуктом

Головне меню системи доступу до хмарного середовища має наступний вигляд, тут відображена коренева папка та основний функціонал системи (Рисунок 5.2.1).

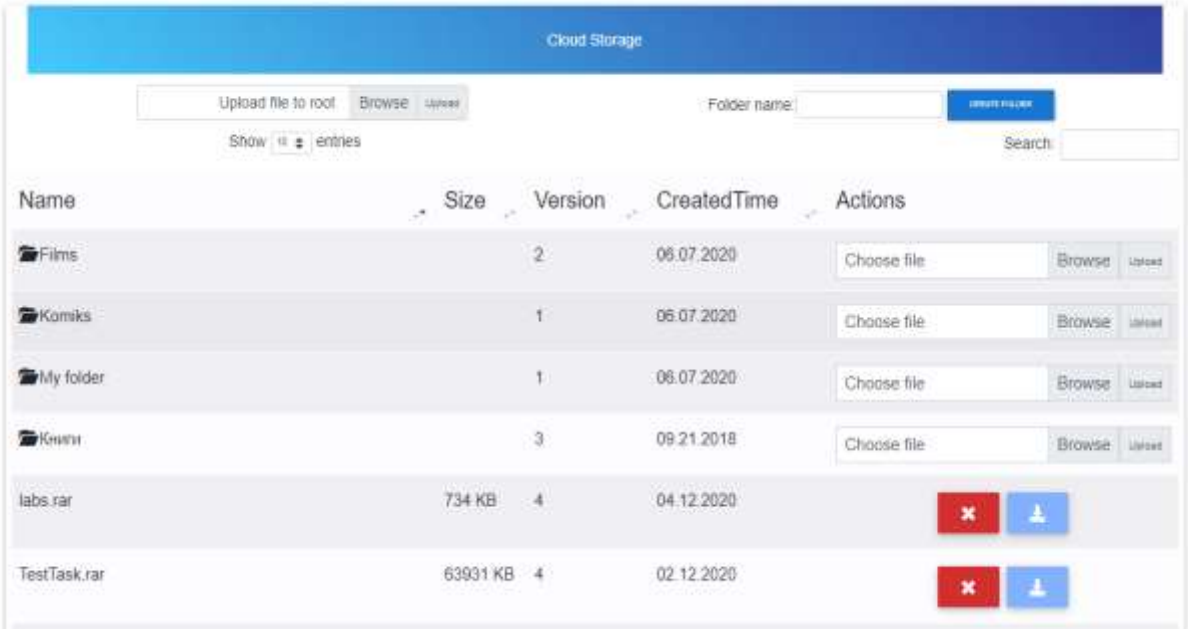

Рисунок 5.2.1 - Коренева папка хмарного сховища

Тут користувач має такі можливості:

- Перегляд директорій, файлів, папок і т. д які існують на Google Disk.
- Пошук по ключовому рядку (пошук ведеться по всім колонкам).
- Сторінкова навігація, з можливістю вибрати кількість об'єктів на сторінці .
- Додавання файлів в будь-яку папку.
- Додавання папок в кореневу або інші.
- Видалення та завантаження файлів.

Меню усіх папок, окрім кореневої має такий вигляд (Рисунок 5.2.2):

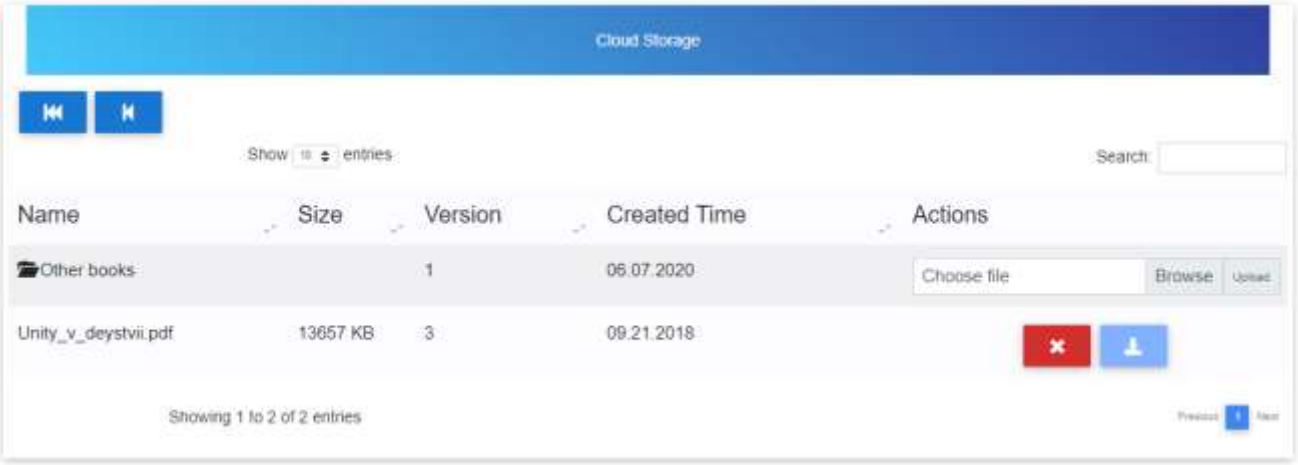

Рисунок 5.2.2 - Папка хмарного сховища

Окрім зазначених вище можливостей в папці окрім кореневої є можливість повернутись до «батьківської» папки або відразу до кореневої.

5.3 Висновки

Розроблена система має функціонал для роботи з файловою системою. Сервіс для обробки запитів працює досить швидко, тому відгук на запит виконується миттєво. Інтерфейс користувача збудований за допомогою стилів Bootstrap 4 через це, він є адаптивним до екранів будь-яких розмірів за винятком 400px та менше.

6 ВИСНОВОК

В ході виконання даної роботи було розроблено програмний засіб доступу та редагування реєстру хмарного середовища.

Програма була написана на мові програмування C# з використанням графічного веб-інтерфейсу.

Програму (модуль) можна використовувати як компонент до інших програм (проектів) без зміни коду.

В ході роботи було проведено огляд та зроблено аналіз засобів, що були використані для створення даного програмного забезпечення (середовища розробки Microsoft Visual Studio 2019, ASP.NET MVC framework та засобів створення веб-інтерфейсів: HTML5, CSS, JavaScript та Bootstrap 4).

Для роботи з даним програмним забезпеченням необхідний лише комп’ютер мінімальної потужності (для запуску браузера).

Користувач має змогу досить швидко додавати, редагувати, видаляти та завантажувати файли з та в хмарне середовище без використання сторонніх ресурсів.

СПИСОК ДЖЕРЕЛ

1. Платформа .NET [Електронний ресурс] — https://metanit.com/sharp/mvc5/

2. C# 6.0 Справочник, полное описание языка, Джозеф Албахари и Бен Албахари

3. Современный учебник CSS [Електронний ресурс] — Режим доступу: https://idg.net.ua/blog/uchebnik-css

4. Сайт Metanit all about C# [Електронний ресурс] — Режим доступу: https://metanit.com/sharp/general.php

5. Google Drive Api .NET Quickstart Complete the steps described in the rest of this page to create a simple .NET console application that makes requests to the Drive API. [Електронний ресурс] — Режим доступу: https://developers.google.com/drive/api/v3/quickstart/dotnet .

6. Tutorial for create API with ASP.NET MVC [Електронний ресурс] — Режим доступу: https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/?view=aspnetcore-3.1.

7. All about SOLID pattern [Електронний ресурс] — Режим доступу: https://habr.com/ru/post/348286/

8. Build fast responsive sites with Bootstrap. [Електронний ресурс] — Режим доступу: https://getbootstrap.com/docs/4.5/getting-started/introduction/

9. OAuth protocol [Електронний ресурс] — Режим доступу: https://developers.google.com/identity/protocols/oauth2

10. Issues with cloud storage [Електронний ресурс] — Режим доступу: https://www.datapine.com/blog/cloud-computing-risks-and-challenges/

11. Material design for Bootstrap 4 [Електронний ресурс] — Режим доступу: https://mdbootstrap.com/education/bootstrap/

12. Library Jquery [Електронний ресурс] — Режим доступу: https://jquery.com/

ДОДАТОК 1

Система доступу до хмарного середовища розподілених електронних ресурсів

Специфікація

УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТМ52124_19Б

Аркушів 2

Київ – 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ52124_19Б	Записка.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ61	CloudStorages/GoogleCloudStorageService.cs	Сервіс додатка
УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ61	CloudStorages\Controllers	Rest full API додатка

ДОДАТОК 2

Система доступу до хмарного середовища розподілених електронних ресурсів

Текст програми

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ61

Аркушів 7

Київ – 2019

```
//interface for CloudStorageService
using CloudStorages.Models;
using Google.Apis.Drive.v3;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Web;

namespace CloudStorages.Interfaces
{
    public interface ICloudStorageService
    {
        List<DriveFile> GetFiles();
        void UploadFile(HttpPostedFileBase file);
        string DownloadFile(string fileId);
        void DeleteFile(DriveFile file);
        List<DriveFile> GetContainsInFolder(String folderId);
        string GetParentFolder(string currentFolderId);
        void CreateFolder(string FolderName);
        void FileUploadInFolder(string folderId, HttpPostedFileBase file);
    }
}

//Model for different entity
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace CloudStorages.Models
{
    public class DriveFile
    {
        public string Id { get; set; }
        public string Name { get; set; }
        public long? Size { get; set; }
        public long? Version { get; set; }
        public DateTime? CreatedTime { get; set; }
        public IList<string> Parents { get; set; }
    }
}
//Controler
using CloudStorages.Interfaces;
using CloudStorages.Models;
using CloudStorages.Services;
using Ninject;
using System.IO;
using System.Web;
using System.Web.Mvc;

namespace CloudStorages.Controllers
{
    public class GoogleController : Controller
    {
        private readonly ICloudStorageService _cloudStorageService;
        public GoogleController(ICloudStorageService cloudStorageService) {
            _cloudStorageService = cloudStorageService;
        }
        [HttpGet]
        public ActionResult GetDriveFiles() {
```

```
            return View("RootFolder",_cloudStorageService.GetFiles());
        }

        [HttpPost]
        public ActionResult DeleteFile(DriveFile file) {
            _cloudStorageService.DeleteFile(file);
            return RedirectToAction("GetDriveFiles");
        }

        [HttpPost]
        public ActionResult UploadFile(HttpPostedFileBase file) {
            _cloudStorageService.UploadFile(file);
            return RedirectToAction("GetDriveFiles");
        }

        public void DownloadFile(string fileId) {
            string FilePath = _cloudStorageService.DownloadFile(fileId);


            Response.ContentType = "application/zip";
            Response.AddHeader("Content-Disposition", "attachment; filename=" +
Path.GetFileName(FilePath));
            Response.WriteFile(System.Web.HttpContext.Current.Server.MapPath("~/DriveFiles/" +
Path.GetFileName(FilePath)));
            Response.End();
            Response.Flush();
        }

        [HttpGet]
        public ActionResult GetContainsInFolder(string folderId)
        {
            //if root
            if(folderId == "0AO5hkdyTHaUeUk9PVA")
            {
                return RedirectToAction("GetDriveFiles");
            }
            return View("Folder",_cloudStorageService.GetContainsInFolder(folderId));
        }

        public ActionResult BackwardPrevioslyFolder(string folderId)
        {
            string parentFolder = _cloudStorageService.GetParentFolder(folderId);
            if (string.IsNullOrEmpty(parentFolder))
            {
                return RedirectToAction("GetDriveFiles");
            }
            return RedirectToAction("GetContainsInFolder", new { folderId = parentFolder });
        }
        [HttpPost]
        public ActionResult CreateFolder(string FolderName)
        {
            _cloudStorageService.CreateFolder(FolderName);
            return RedirectToAction("GetDriveFiles");
        }

        [HttpPost]
        public ActionResult FileUploadInFolder(DriveFile FolderId, HttpPostedFileBase file)
        {
            _cloudStorageService.FileUploadInFolder(FolderId.Id, file);
            return RedirectToAction("GetDriveFiles");
        }
    }
}
```

```
//Service
using CloudStorages.Interfaces;
using CloudStorages.Models;
using Google.Apis.Auth.OAuth2;
using Google.Apis.Download;
using GoogleDriveV2 = Google.Apis.Drive.v2;
using Google.Apis.Drive.v3;
using Google.Apis.Services;
using Google.Apis.Util.Store;
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading;
using System.Web;
using System.Linq;

namespace CloudStorages.Services
{
    public class GoogleCloudStorageService : ICloudStorageService
    {
        private string[] Scopes = { DriveService.Scope.Drive };

        private DriveService GetService()
        {
            UserCredential credential;
            using (var stream = new FileStream(@"C:\Main\Diplom\client_secret.json",
FileMode.Open, FileAccess.Read))
            {
                String FolderPath = @"C:\Main\Diplom\";
                String FilePath = Path.Combine(FolderPath, "DriveServiceCredentials.json");

                credential = GoogleWebAuthorizationBroker.AuthorizeAsync(
                    GoogleClientSecrets.Load(stream).Secrets,
                    Scopes,
                    "user",
                    CancellationToken.None,
                    new FileDataStore(FilePath, true)).Result;
            }

            //Create Drive API service.
            DriveService service = new DriveService(new BaseClientService.Initializer()
            {
                HttpClientInitializer = credential,
                ApplicationName = "CloudStorages",
            });

            return service;
        }

        private GoogleDriveV2.DriveService GetServiceV2()
        {
            UserCredential credential;
            using (var stream = new FileStream(@"C:\Main\Diplom\client_secret.json",
FileMode.Open, FileAccess.Read))
            {
                String FolderPath = @"C:\Main\Diplom\";
                String FilePath = Path.Combine(FolderPath, "DriveServiceCredentials.json");

                credential = GoogleWebAuthorizationBroker.AuthorizeAsync(
                    GoogleClientSecrets.Load(stream).Secrets,
                    Scopes,
                    "user",
                    CancellationToken.None,
```

```
                    new FileDataStore(FilePath, true)).Result;
            }

            //Create Drive API service.
            GoogleDriveV2.DriveService service = new GoogleDriveV2.DriveService(new
BaseClientService.Initializer()
            {
                HttpClientInitializer = credential,
                ApplicationName = "GoogleDriveRestAPI-v2",
            });

            return service;
        }

        public  List<DriveFile> GetFiles()
        {
            DriveService service = GetService();

            // Define parameters of request.
            FilesResource.ListRequest FileListRequest = service.Files.List();

            //listRequest.PageSize = 10;
            //listRequest.PageToken = 10;
            FileListRequest.Fields = "nextPageToken, files(id, name, size, version, trashed,
createdTime, parents)";

            // List files.
            IList<Google.Apis.Drive.v3.Data.File> files = FileListRequest.Execute().Files;
            List<DriveFile> FileList = new List<DriveFile>();

            if (files != null && files.Count > 0)
            {
                foreach (var file in files)
                {
                    DriveFile File = new DriveFile {
                        Id = file.Id,
                        Name = file.Name,
                        Size = file.Size,
                        Version = file.Version,
                        CreatedTime = file.CreatedTime,
                        Parents = file.Parents
                    };
                    FileList.Add(File);
                }
            }
            return FileList.Where(f => f.Version != 5).ToList();
        }

        public void UploadFile(HttpPostedFileBase file)
        {
            if (file != null && file.ContentLength > 0)
            {
                DriveService service = GetService();

                string path = Path.Combine(HttpContext.Current.Server.MapPath("~/DriveFiles"),
                Path.GetFileName(file.FileName));
                file.SaveAs(path);

                var FileMetaData = new Google.Apis.Drive.v3.Data.File();
                FileMetaData.Name = Path.GetFileName(file.FileName);
                FileMetaData.MimeType = MimeMapping.GetMimeMapping(path);

                FilesResource.CreateMediaUpload request;
```

```
                using (var stream = new System.IO.FileStream(path, System.IO.FileMode.Open))
                {
                    request = service.Files.Create(FileMetaData, stream, FileMetaData.MimeType);
                    request.Fields = "id";
                    request.Upload();
                }
            }
        }

        public string DownloadFile(string fileId)
        {
            DriveService service = GetService();

            string FolderPath = System.Web.HttpContext.Current.Server.MapPath("/DriveFiles/");
            FilesResource.GetRequest request = service.Files.Get(fileId);

            string FileName = request.Execute().Name;
            string FilePath = System.IO.Path.Combine(FolderPath, FileName);

            MemoryStream stream1 = new MemoryStream();

            request.MediaDownloader.ProgressChanged += (Google.Apis.Download.IDownloadProgress
progress) =>
            {
                switch (progress.Status)
                {
                    case DownloadStatus.Downloading:
                        {
                            break;
                        }
                    case DownloadStatus.Completed:
                        {
                            SaveStream(stream1, FilePath);
                            break;
                        }
                    case DownloadStatus.Failed:
                        {
                            break;
                        }
                }
            };
            request.Download(stream1);
            return FilePath;
        }

        private void SaveStream(MemoryStream stream, string FilePath)
        {
            using (System.IO.FileStream file = new FileStream(FilePath, FileMode.Create,
FileAccess.ReadWrite))
            {
                stream.WriteTo(file);
            }
        }

        public void DeleteFile(DriveFile file)
        {
            DriveService service = GetService();
            try
            {
                // Initial validation.
                if (service == null)
                    throw new ArgumentNullException("service");
```

```
                if (file.Id == null)
                    throw new ArgumentNullException(file.Id);

                // Make the request.
                service.Files.Delete(file.Id).Execute();
            }
            catch (Exception ex)
            {
                throw new Exception("Request Files.Delete failed.", ex);
            }
        }
        public List<DriveFile> GetContainsInFolder(String folderId)
        {
            List<string> ChildList = new List<string>();
            var serviceV2 = GetServiceV2();
            GoogleDriveV2.ChildrenResource.ListRequest ChildrenIDsRequest =
serviceV2.Children.List(folderId);
            do
            {
                var children = ChildrenIDsRequest.Execute();

                if (children.Items != null && children.Items.Count > 0)
                {
                    foreach (var file in children.Items)
                    {
                        ChildList.Add(file.Id);
                    }
                }
                ChildrenIDsRequest.PageToken = children.NextPageToken;

            } while (!String.IsNullOrEmpty(ChildrenIDsRequest.PageToken));

            //Get All File List
            List<DriveFile> AllFileList = GetFiles();
            List<DriveFile> Filter_FileList = new List<DriveFile>();

            foreach (string Id in ChildList)
            {
                Filter_FileList.Add(AllFileList.Where(x => x.Id == Id).FirstOrDefault());
            }

            return Filter_FileList;
        }

        public string GetParentFolder(string currentFolderId)
        {
            List<DriveFile> AllFileList = GetFiles();
            return AllFileList.FirstOrDefault(f => f.Id == currentFolderId)?.Parents[0];
        }
        public void CreateFolder(string FolderName)
        {
            Google.Apis.Drive.v3.DriveService service = GetService();

            Google.Apis.Drive.v3.Data.File FileMetaData = new Google.Apis.Drive.v3.Data.File();
            FileMetaData.Name = FolderName;
            FileMetaData.MimeType = "application/vnd.google-apps.folder";

            Google.Apis.Drive.v3.FilesResource.CreateRequest request;

            request = service.Files.Create(FileMetaData);
            request.Fields = "id";
            var file = request.Execute();
```

```
            //Console.WriteLine("Folder ID: " + file.Id);
        }
        public void FileUploadInFolder(string folderId, HttpPostedFileBase file)
        {
            if (file != null && file.ContentLength > 0)
            {
                DriveService service = GetService();

                string path = Path.Combine(HttpContext.Current.Server.MapPath("~/DriveFiles"),
                Path.GetFileName(file.FileName));
                file.SaveAs(path);

                var FileMetaData = new Google.Apis.Drive.v3.Data.File()
                {
                    Name = Path.GetFileName(file.FileName),
                    MimeType = MimeMapping.GetMimeMapping(path),
                    Parents = new List<string>
                    {
                        folderId
                    }
                };

                Google.Apis.Drive.v3.FilesResource.CreateMediaUpload request;
                using (var stream = new System.IO.FileStream(path, System.IO.FileMode.Open))
                {
                    request = service.Files.Create(FileMetaData, stream, FileMetaData.MimeType);
                    request.Fields = "id";
                    request.Upload();
                }
                var file1 = request.ResponseBody;
            }
        }
    }
}
```

ДОДАТОК З

Система доступу до хмарного середовища розподілених електронних ресурсів

Опис програми

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТМ61

Аркушів 8

Київ – 2019

АНОТАЦІЯ

Метою розробки системи доступу до хмарних середовищ розподілених електронних ресурсів є створення програмного продукту, що надає простий інтерфейс користувачу для роботи з хмарним середовищем та в майбутньому використовувати як модуль будь-якого проекту, при невеликій кастомизації.

Система має простий, адаптивний інтерфейс та швидку обробку запитів клієнта.

ЗМІСТ

-4-

ЗАГАЛЬНІ ВІДОМОСТІ

Відповідно до назви дипломної роботи, розроблений продукт система доступу до хмарного середовища розподілених електронних ресурсів. Система працює як веб-служба при розвертанні на IIS.

Користувач для роботи з системою повинен мати аккаунт Google та зареєструвати програму в Google Api Concole для надання доступу до Google Disk.

-5-

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Система повинна моделювати файлову систему, тому вона виконує такі функції:

— Перегляд директорій, файлів, папок і т. д які існують на хмарному середовищі.

— Пошук по ключовому рядку (пошук ведеться по всім колонкам).

— Сторінкова навігація, з можливістю вибрати кількість об'єктів на сторінці .

— Додавання файлів в будь-яку папку.

— Додавання папок в кореневу або інші.

— Видалення та завантаження файлів.

-6-

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Загальний принцип роботи додатку такий. Користувач переглядає директорії, файли, папки і т.д., редагує та завантажує локально реєстр.

-7-

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для користування даною системою потрібен браузер, який підтримує нові каскадні стилі Bootstrap 4.

-8-

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними є:

— облікові дані користувача;

— дії користувача;

Вихідними даними є:

— реєстр хмарного середовища;